Opt-AI.

Weekly Seminar

Opt-AI. LLM Research Team

Executorch QNN Quantization

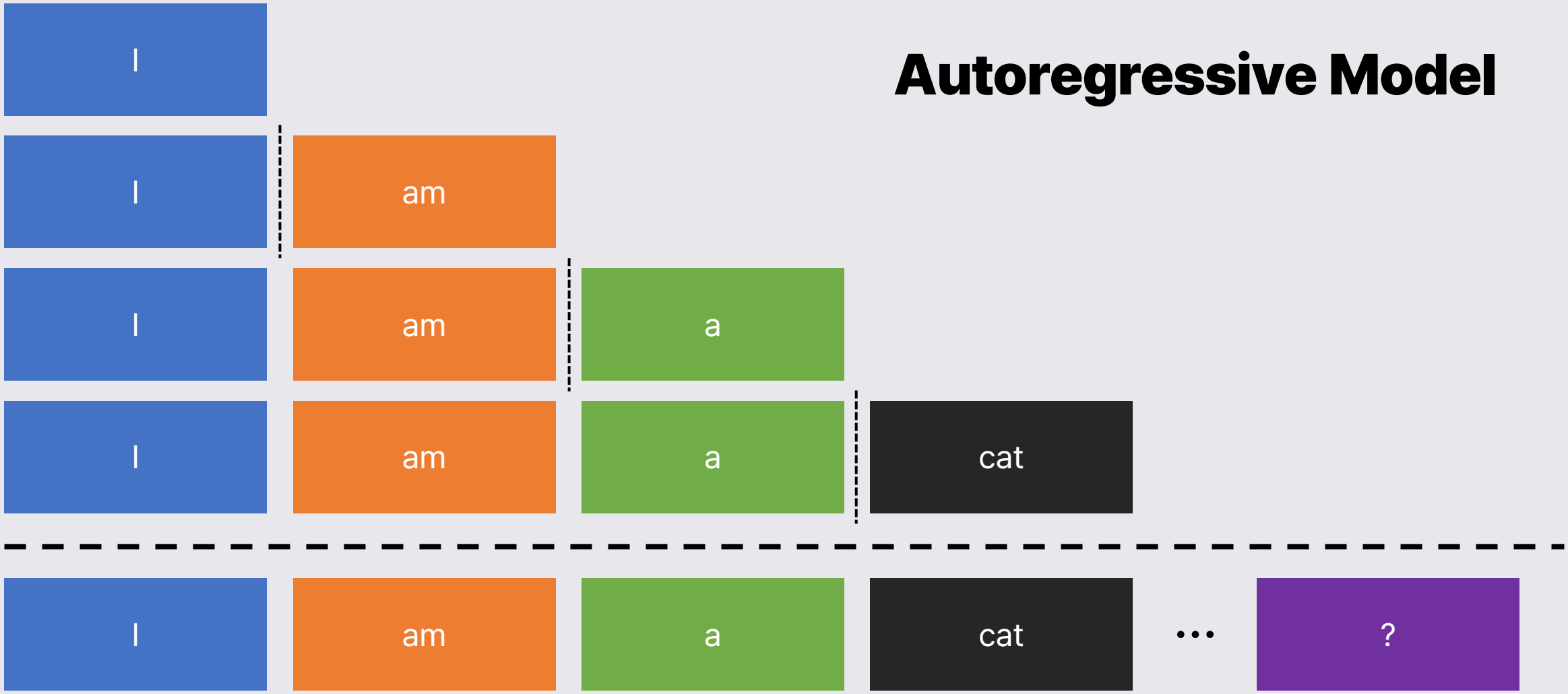
Table of Contents

01

LLM KV-Cache

LLM KV-Cache

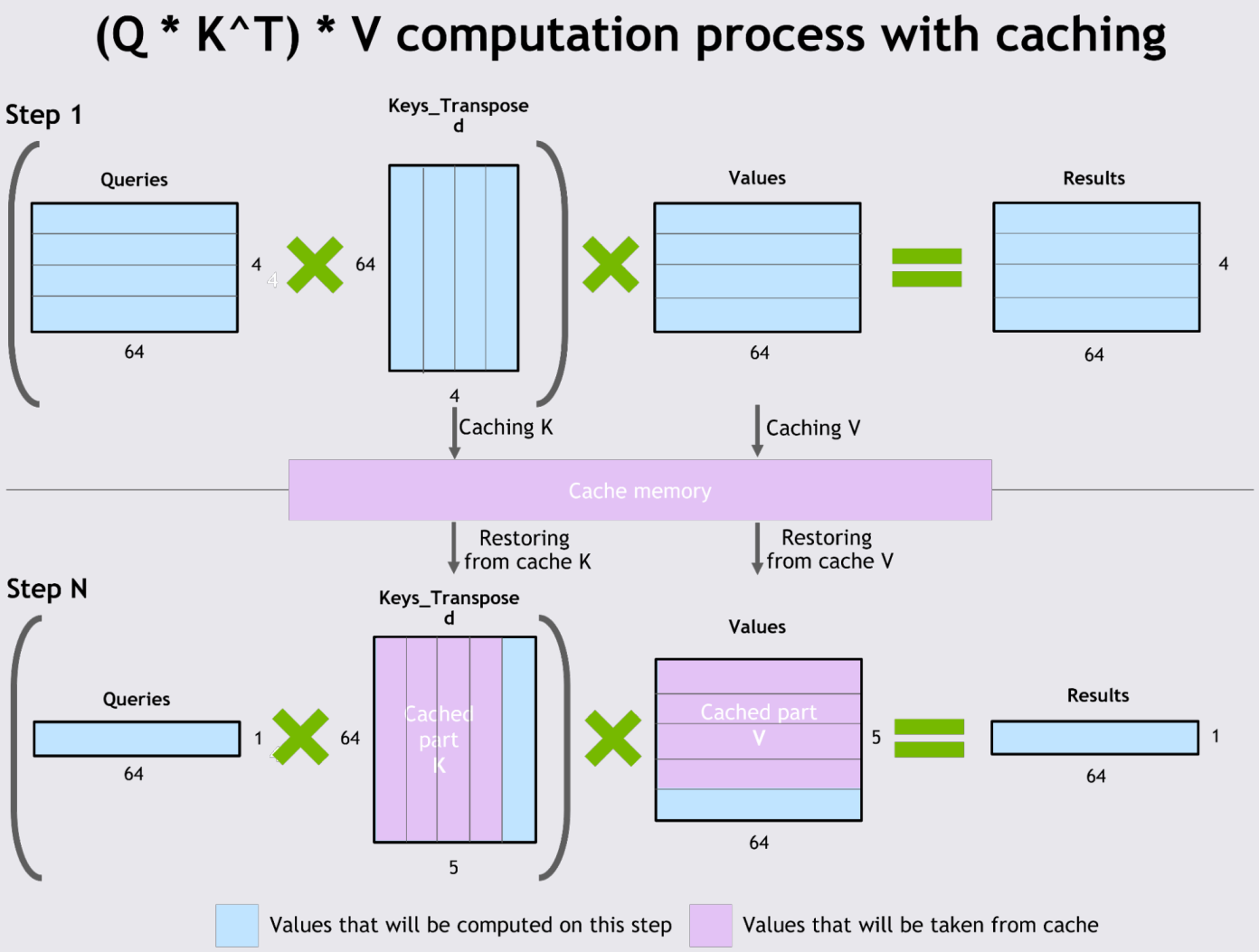
What is KV-Cache

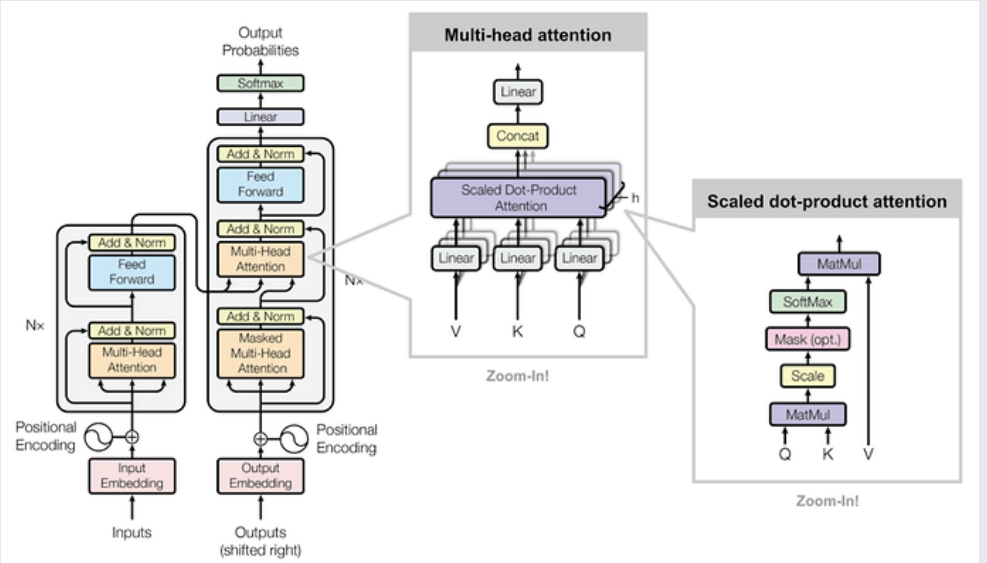


Autoregressive Model

LLM KV-Cache

What is KV-Cache

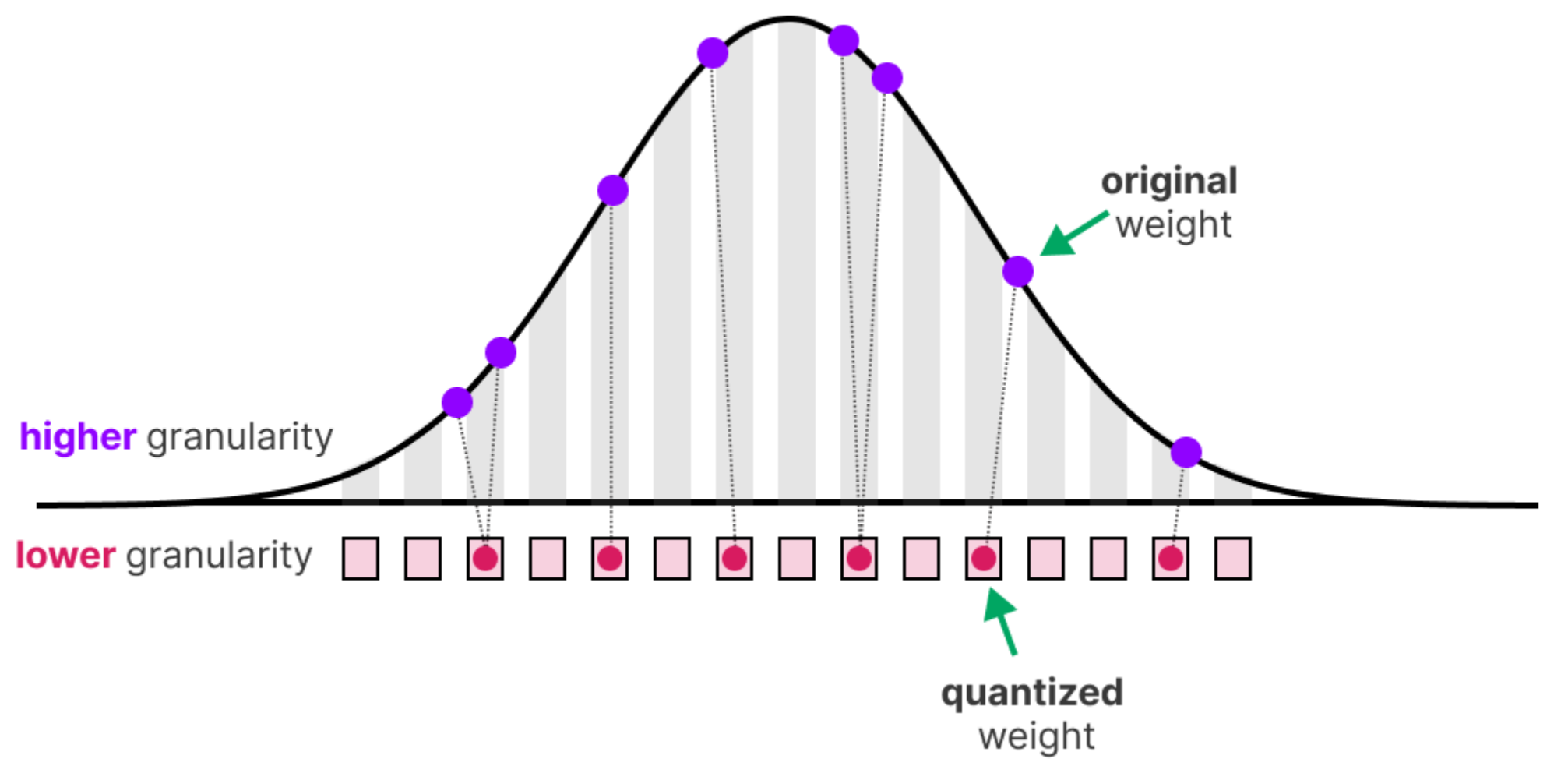




02
LLM Quantization

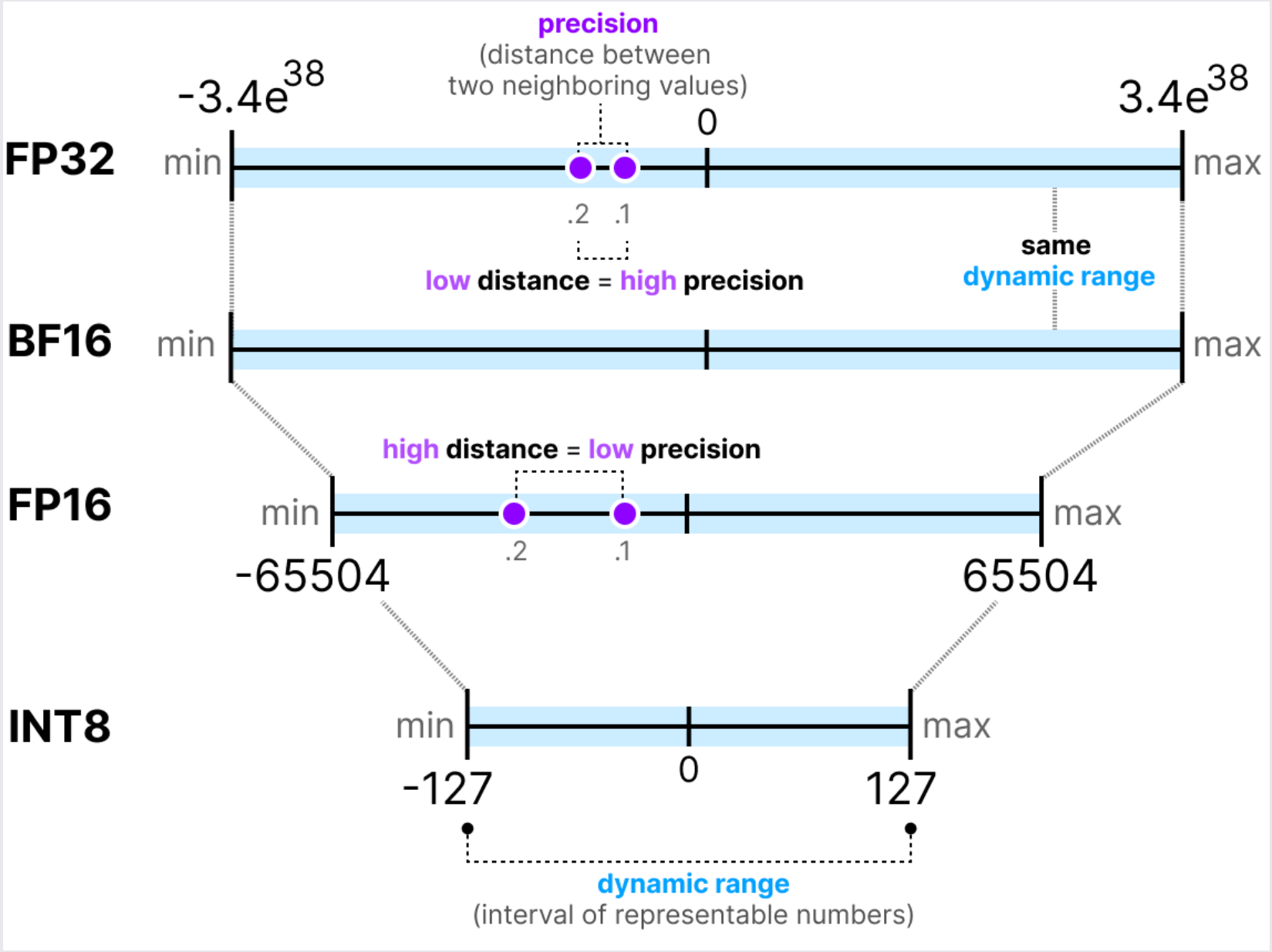
LLM Quantization

Quantization



LLM Quantization

Data Type



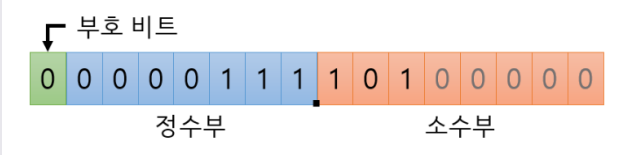
LLM Quantization

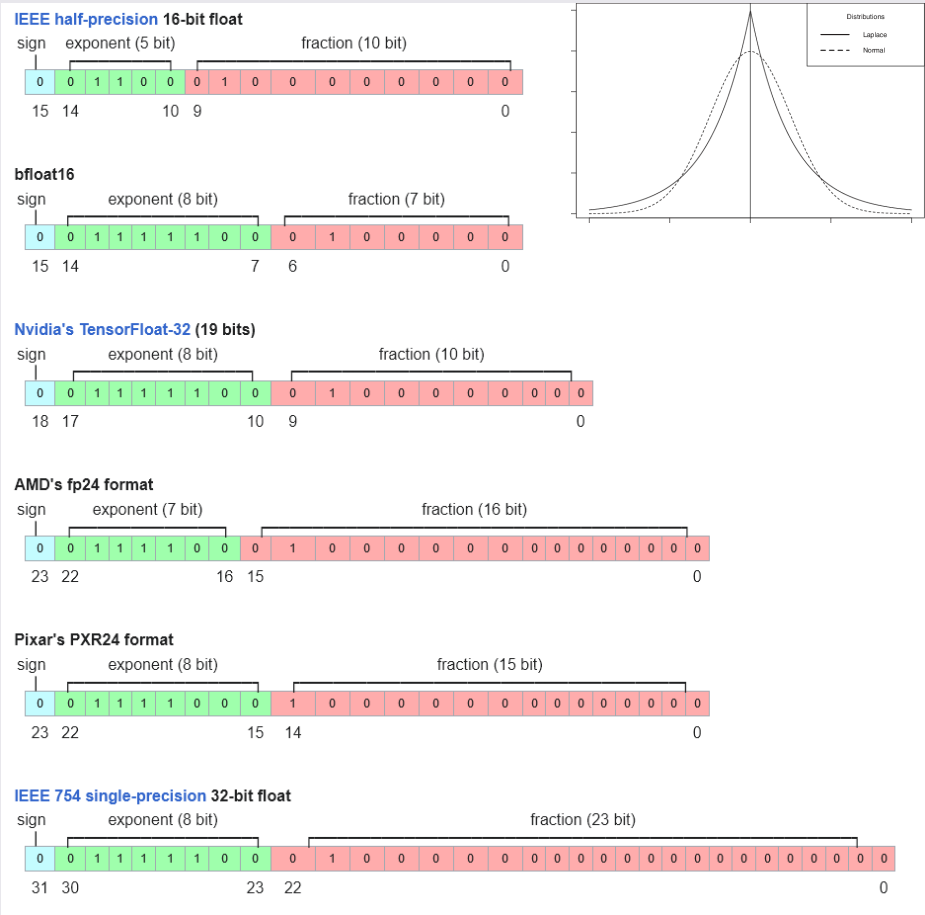
Data Type

FIXED POINT vs FLOATING POINT

- FIXED POINT
  - 정수부/소수부로 나누어 저장
  - 기존 INTEGER data format과 높은 호환성을 가지고 있어 하드웨어 친화적
  - 정수부/소수부 위치 등은 별도 관리 필요
  - Uniform distribution 또는 Non-zero mean data(Affine Quantization)에 적합

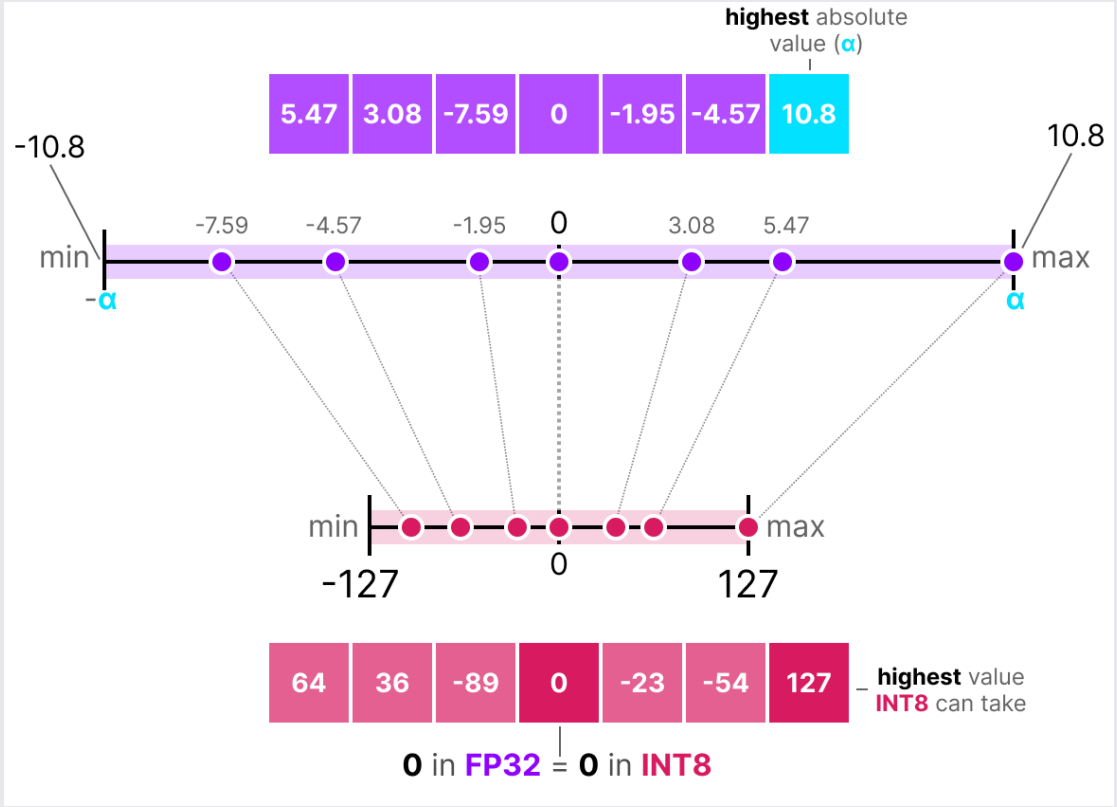
- FLOATING POINT
  - 지수부/소수부로 나누어 저장 (0.0078125 → 1.0 * 2^-6)
  - INTEGER 대비 하드웨어 구조 복잡
  - Laplace distribution에 적합





Symmetric Quantization

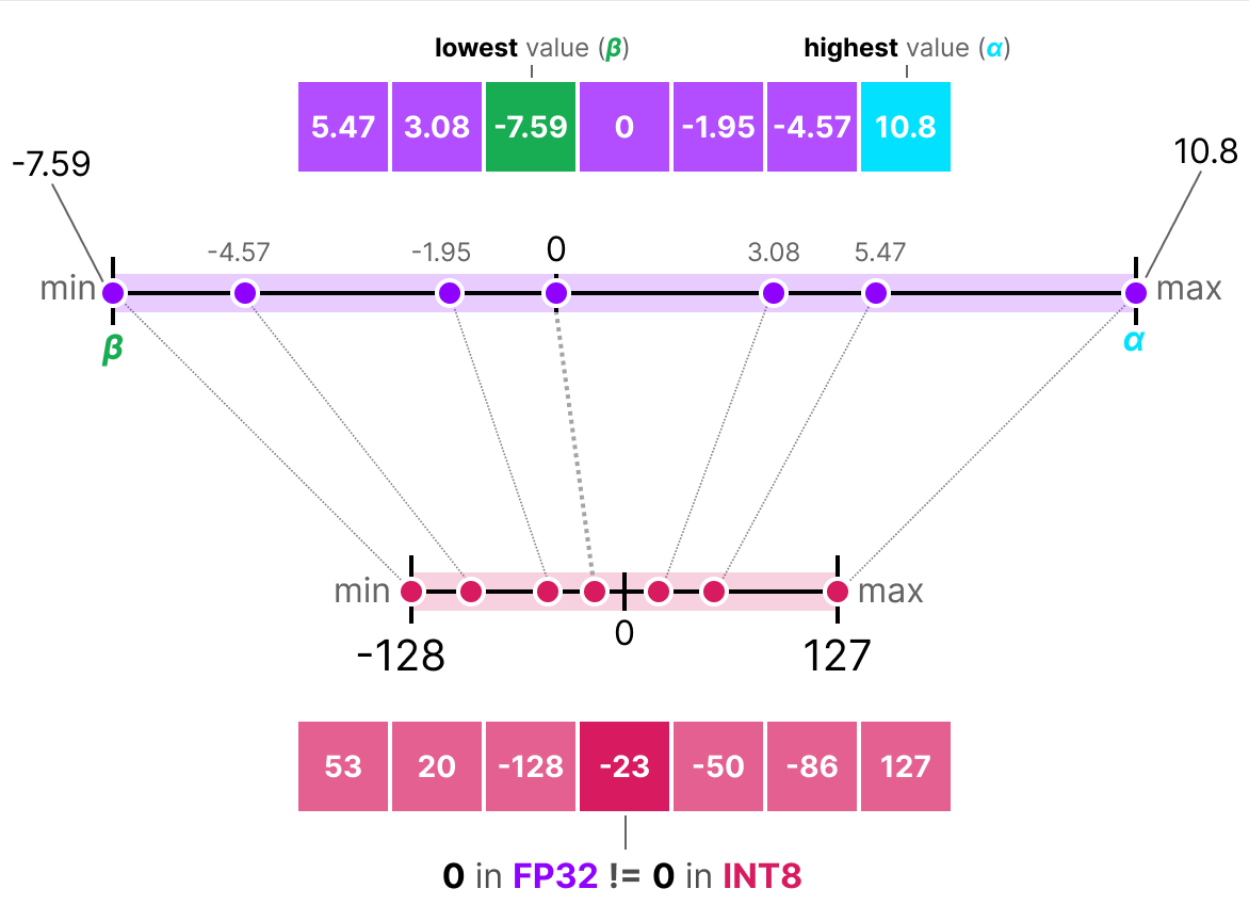
- 0을 중심으로 대칭적인 정수 범위에 매핑하는 방식
- Zero-point가 항상 0으로 실수 값이 0이라면 정수 값도 항상 0을 가짐
- 실수 값의 최대, 최소의 크기가 대칭적이라고 가정



$$Q_x = round(\frac{X}{S})$$

Affine Quantization

- 입력 값을 Affine Transformation하여 정수 값으로 변환 하는 방식
- Affine Quantization은 Asymmetric Quantization 양자화를 포함하고 있음
- 실수 값이 비대칭적



$$Q_x = round(\frac{X}{S} + Z)$$

$$X = S \cdot (Q_x - Z)$$

Q_x: 양자화된 정수 값
X: 원래 실수 값,
S: 실수 값과 정수 값 간의 단위 변환
Z: 실수 값에서 0을 정수 값으로 변환하기 위한 값

LLM Quantization

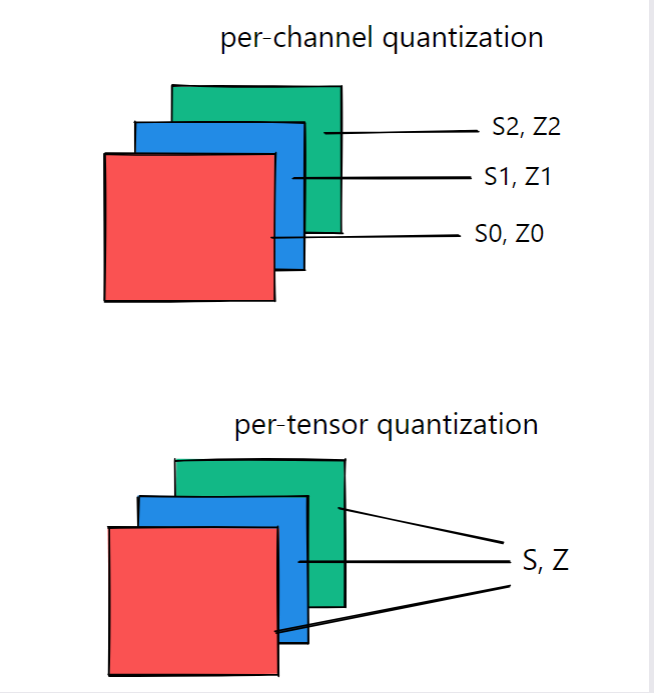
Per-Tensor/Channel Quantization

Per-Channel Quantization

- Tensor의 각 채널마다 독립적인 Scaling factor와 Zero-point 사용해 양자화
- 각 채널의 데이터 분포에 맞춘 Scaling과 Zero-point를 사용하므로 데이터 분포의 차이로 인한 정보 손실 줄임
- 채널마다 별도의 Scaling factor와 Zero-point 가지므로, 메모리 요구량 증가

Per-Tensor Quantization

- Tensor의 모든 값이 동일한 Scaling factor와 Zero-point를 사용해 양자화
- 계산이 단순하며 메모리 요구량이 낮음
- Tensor 내 데이터의 분포가 채널마다 다를 경우, 정확도 손실



LLM Quantization

Calibration

Calibration

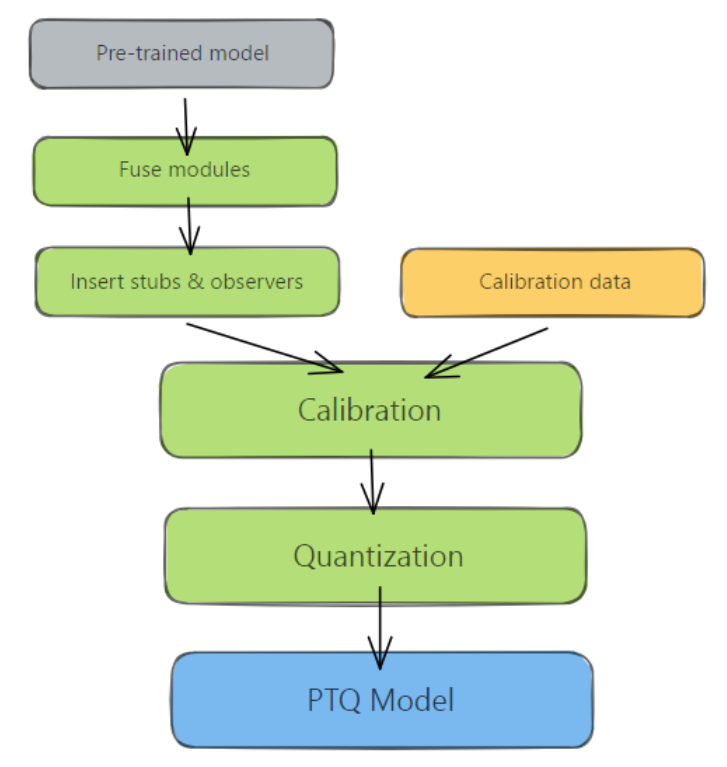
- 양자화를 수행할 [a, b] 범위 설정을 위한 방법
- 실수 값을 정수 값으로 변환하기 전, 데이터의 범위를 분석하고 스케일링을 설정하는 과정

Post-Training Dynamic Quantization

- PTQ의 일환으로, 모델을 훈련한 후 양자화를 적용하는 대표적인 방법
- 모델 Weight는 고정된 정수 값으로 양자화, Activation은 Runtime중에 동적으로 변환

Post-Training Static Quantization

- Weight와 Activation을 모두 정적인 범위를 기반으로 양자화
- Activation 값의 범위를 추정하기 위해 Calibration 데이터 사용

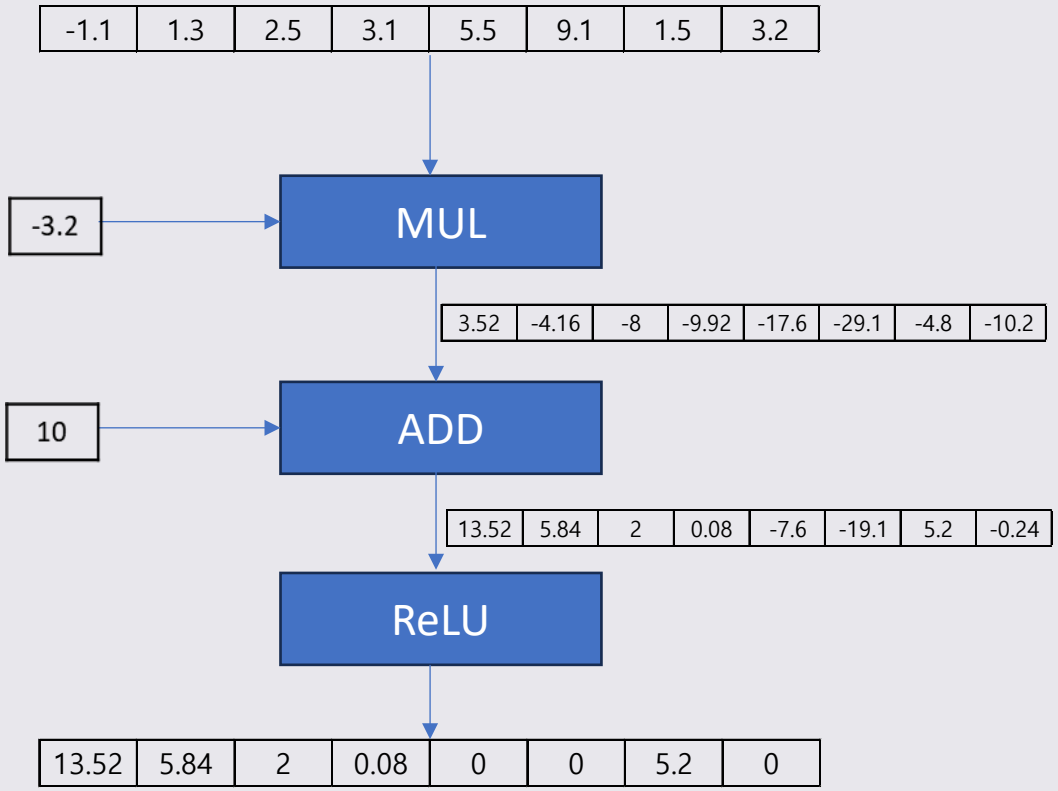


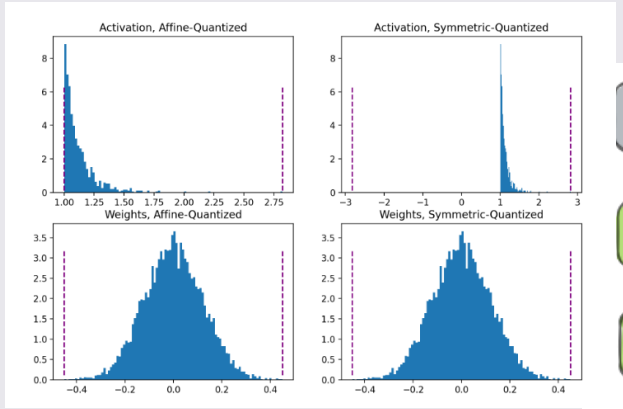
LLM Quantization

Calibration

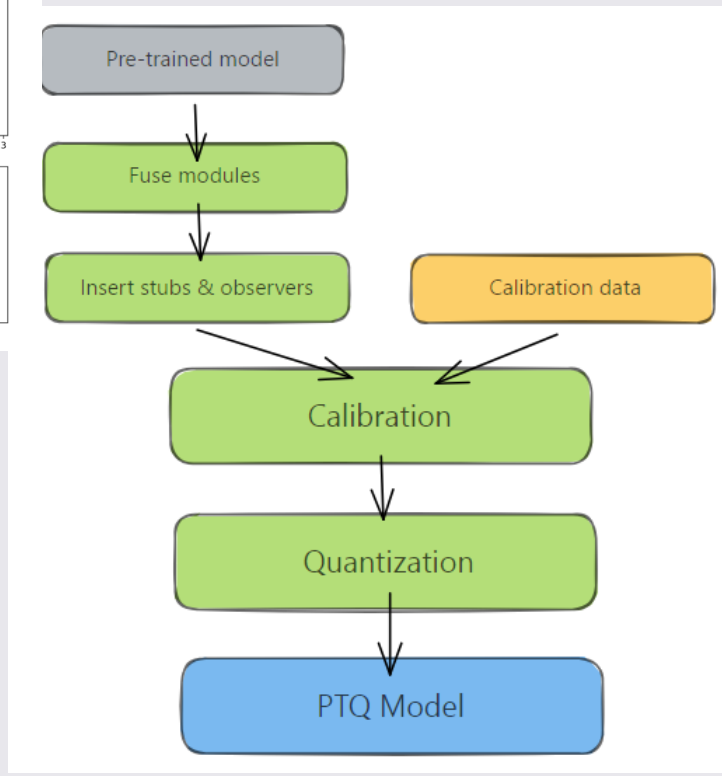
Calibration

- Observer



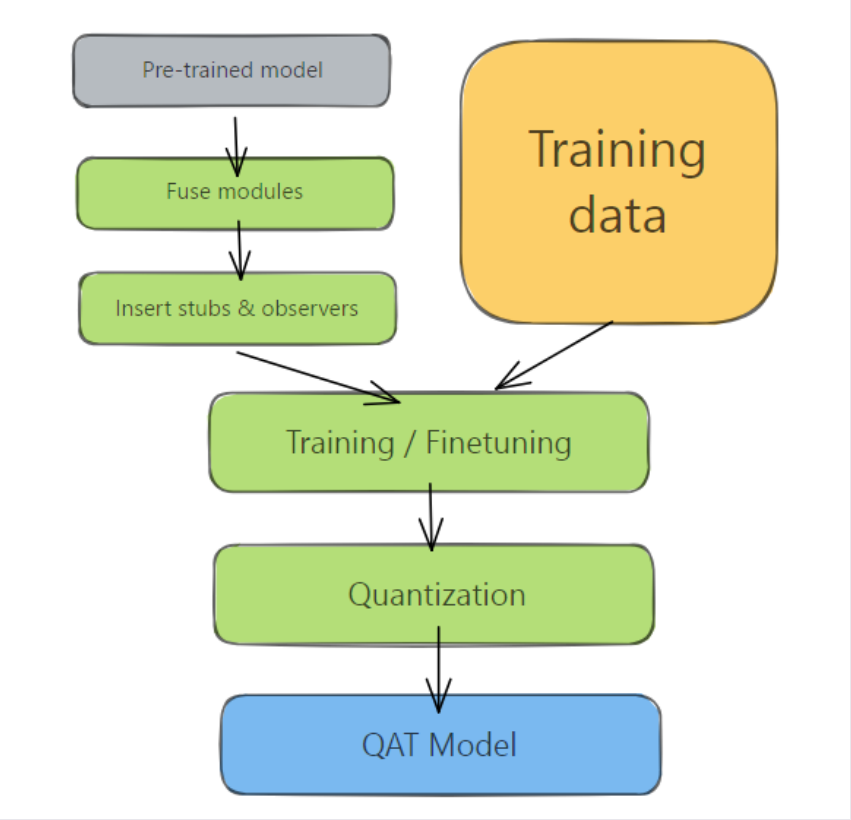


Min	-19.1
Max	13.52
Median	1.04
Std	9.793



Quantization-Aware Training (QAT)

- 훈련 과정에서 양자화를 시뮬레이션하며, 모델을 양자화 친화적으로 학습
- 훈련 중 Weight와 Activation을 모두 양자화된 상태로 처리
- 양자화로 발생하는 손실에 모델이 적응할 수 있도록 함
- 정확도는 가장 높을 수 있으나, 훈련 비용과 계산 비용 증가



Min-Max

- 최대값과 최소값을 기준으로 범위 설정 (Weight)
- 간단하고 빠르나, 이상치에 민감

Moving Average Min-Max

- 샘플 데이터의 평균적인 분포를 기준으로 범위 설정 (Activation)
- 이상치 영향을 줄이고, 안정적인 스케일링 제공

03
Executorch QNN

Executorch QNN

QNN Quantization

```
parser.add_argument(
        "--qnn",
        action="store_true",
        help="Delegate llama2 to qnn backend (Qualcomm), please use it --kv_cahce=True",
    )
```

```
parser.add_argument(
        "--pt2e_quantize",
        default=None,
        choices=[
            "xnnpack_dynamic",
            "xnnpack_dynamic_qc4",
            "qnn_8a8w",
            "qnn_16a16w",
            "qnn_16a4w",
            "coreml_c4w",
            "coreml_8a_c8w",
            "coreml_8a_c4w",
            "coreml_baseline_8a_c8w",
            "coreml_baseline_8a_c4w",
            "vulkan_8w",
        ],
    )
```

Executorch QNN

QNN Quantization

```python
def _export_llama(modelname, args) -> LLMEdgeManager:  # noqa: C901
    _validate_args(args)
    pt2e_quant_params, quantizers, quant_dtype = get_quantizer_and_quant_params(args)

    # export_to_edge
    builder_exported_to_edge = (
        _prepare_for_llama_export(modelname, args)
        .export()
        .pt2e_quantize(quantizers)
        .export_to_edge(
    )
```

```python
def get_quantizer_and_quant_params(args):
    pt2e_quant_params = get_pt2e_quantization_params(
        args.pt2e_quantize, args.quantization_mode
    )
    quantizers = get_pt2e_quantizers(pt2e_quant_params, args.so_library)
    quant_dtype = None
    if args.qnn and args.pt2e_quantize:
        assert len(quantizers) == 0, "Should not enable both xnnpack and qnn"
        qnn_quantizer, quant_dtype = get_qnn_quantizer(
            args.pt2e_quantize, args.quantization_mode
        )
        quantizers.append(qnn_quantizer)
```

```
print(args.pt2e_quantize, args.quantization_mode, pt2e_quant_params)
qnn_16a8w None None
```

Executorch QNN

QNN Quantization

```python
def get_qnn_quantizer(
    pt2e_quantize: str,
    quantization_mode: Optional[str] = None,
    is_qat: bool = False,
):

    backend, quant_config = pt2e_quantize.split("_")
    assert (
        backend == "qnn"
    ), f"The quantization config is for backend {backend} instead of qnn."
    qnn_quantizer = QnnQuantizer()  # pyre-fixme[16]
    qnn_quantizer.set_per_channel_conv_quant(enable=True)
    qnn_quantizer.set_per_channel_linear_quant(enable=True)
    # more custom quantization are supported including 16a4w etc. default to 8bit quantized
    custom_annotations = ()
    if quant_config == "8a8w":
        quant_dtype = QuantDtype.use_8a8w  # pyre-fixme[16]
        qnn_quantizer.set_quant_config(quant_dtype, is_qat=is_qat)
    elif quant_config == "16a16w":
        quant_dtype = QuantDtype.use_16a16w  # pyre-fixme[16]
        # Due to the error with 16a16w in Qnn Htp, we need to disable per channel linear quantization when use 16a16w
        # TODO: enable it after the issue is fixed
        logging.warning(
            "Disable per channel quantization for linear and conv due to the error with QNN HTP 16a16w."
        )
        qnn_quantizer.set_per_channel_conv_quant(enable=False)
        qnn_quantizer.set_per_channel_linear_quant(enable=False)
        # pyre-ignore: Undefined attribute [16]: Module `executorch.backends` has no attribute `qualcomm`.
        qnn_quantizer.set_quant_config(
            quant_dtype, is_qat=is_qat, act_observer=MinMaxObserver
        )
    elif quant_config == "16a4w":
        # pyre-ignore: Undefined attribute [16]: Module `executorch.backends` has no attribute `qualcomm`.
        quant_dtype = QuantDtype.use_16a4w
        # pyre-ignore: Undefined attribute [16]: Module `executorch.backends` has no attribute `qualcomm`.
        qnn_quantizer.set_quant_config(
            quant_dtype, is_qat=is_qat, act_observer=MinMaxObserver
        )
        # pyre-ignore: Undefined attribute [16]: Module `executorch.backends` has no attribute `qualcomm`.
        custom_annotations = (custom_annotate_llama_matmul_16a8w,)
    else:
        raise AssertionError(
            f"No support for quant type {quant_config}. Support 8a8w, 16a16w and 16a4w."
        )

    assert (
        quantization_mode is None
    ), "Currently qnn backend only supports QnnQuantizer via pt2e flow"
    qnn_quantizer.add_custom_quant_annotations(custom_annotations)

    return qnn_quantizer, quant_dtype
```

Executorch QNN

QNN Quantization

```python
class QnnQuantizer(Quantizer):
    SUPPORTED_OPS: Set = set(OP_ANNOTATOR.keys())

    def __init__(self):
        super().__init__()
        self.quant_ops: Set[OpOverload] = self.SUPPORTED_OPS.copy()

        self.is_qat = False
        self.quant_dtype = QuantDtype.use_8a8w
        self.quant_config: QuantizationConfig = get_8a8w_qnn_ptq_config()
        self.per_channel_quant_config = get_ptq_per_channel_quant_config()
        self.use_per_channel_weight_quant_ops: Set[OpOverload] = set()

        self.custom_quant_annotations: Sequence[Callable] = []
        self.discard_nodes: Set[str] = set()
```

Executorch QNN

QNN Quantization

```python
def set_quant_config(
        self, quant_dtype: QuantDtype, is_qat=False, act_observer=None
    ) -> None:
        self.quant_dtype = quant_dtype
        self.is_qat = is_qat
        if (quant_dtype, is_qat) not in quant_config_dict:
            raise RuntimeError(
                f"the quant config, (quant_dtype: {quant_dtype}, is_qat: {is_qat}) is not support"
            )

        quant_config_fuc, self.per_channel_quant_config = quant_config_dict[
            (quant_dtype, is_qat)
        ]
        self.quant_config = (
            quant_config_fuc(act_observer) if act_observer else quant_config_fuc()
        )
```

```python
quant_config_dict = {
    # PTQ
    (QuantDtype.use_16a16w, False): (
        get_16a16w_qnn_ptq_config,
        get_ptq_per_channel_quant_config(torch.uint16, torch.int16),
    ),
    (QuantDtype.use_16a8w, False): (
        get_16a8w_qnn_ptq_config,
        get_ptq_per_channel_quant_config(torch.uint16, torch.int8),
    ),
    (QuantDtype.use_16a4w, False): (
        get_16a4w_qnn_ptq_config,
        get_ptq_per_channel_quant_config(torch.uint16, "int4"),
    ),
    (QuantDtype.use_8a8w, False): (
        get_8a8w_qnn_ptq_config,
        get_ptq_per_channel_quant_config(),
    ),
    # QAT,
    (QuantDtype.use_16a4w, True): (
        get_16a4w_qnn_qat_config,
        get_qat_per_channel_quant_config(torch.uint16, "int4"),
    ),
    (QuantDtype.use_8a8w, True): (
        get_8a8w_qnn_qat_config,
        get_qat_per_channel_quant_config(),
    ),
}
```

Executorch QNN

QNN Quantization

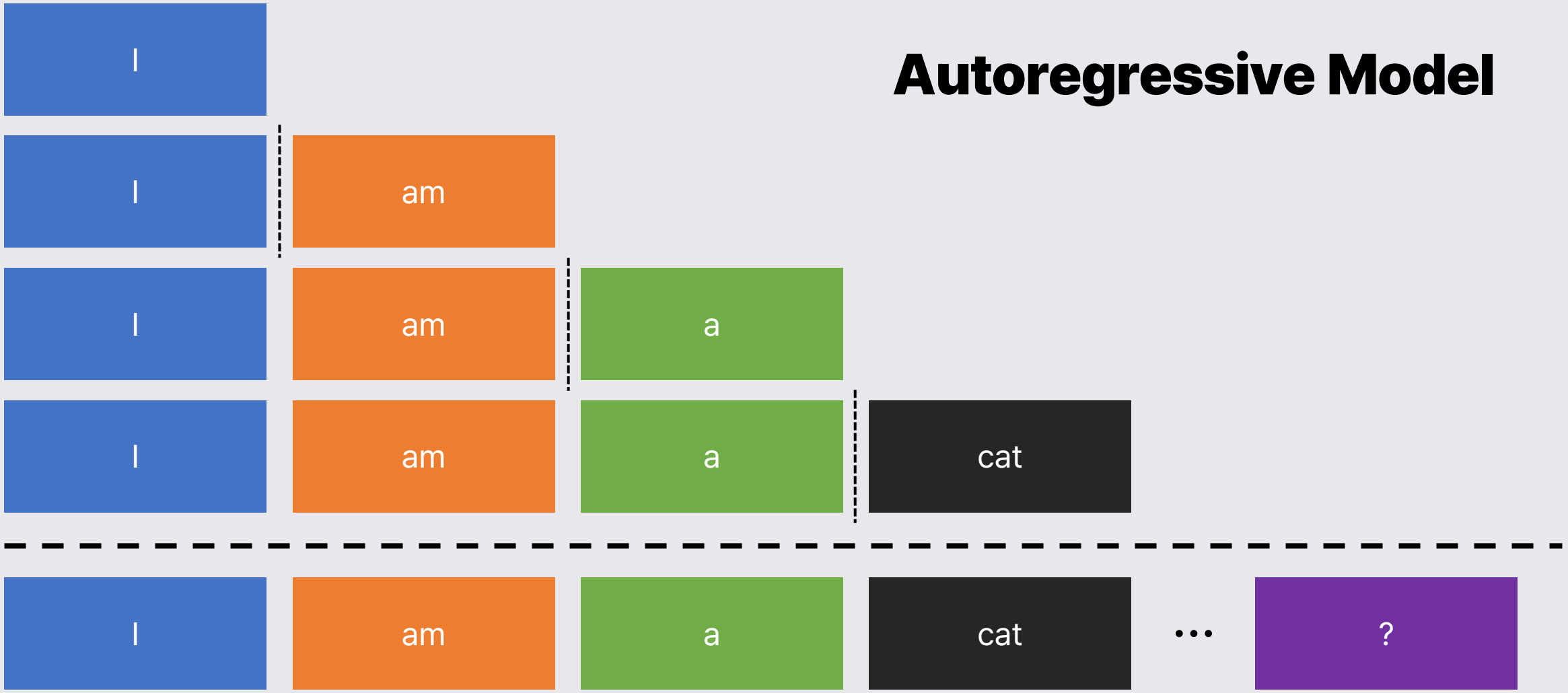
```python
def get_8a8w_qnn_ptq_config(
    act_symmetric: bool = False, act_observer=MovingAverageMinMaxObserver
) -> QuantizationConfig:
    extra_args: Dict[str, Any] = {"eps": 2**-12}

    act_quantization_spec = QuantizationSpec(
        dtype=torch.uint8,
        qscheme=(
            torch.per_tensor_symmetric if act_symmetric else torch.per_tensor_affine
        ),
        ch_axis=0,
        observer_or_fake_quant_ctr=act_observer.with_args(**extra_args),
    )

    weight_quantization_spec = QuantizationSpec(
        dtype=torch.int8,
        quant_min=torch.iinfo(torch.int8).min + 1,
        quant_max=torch.iinfo(torch.int8).max,
        qscheme=torch.per_tensor_symmetric,
        ch_axis=0,
        observer_or_fake_quant_ctr=MinMaxObserver.with_args(**extra_args),
    )

    bias_quantization_spec = QuantizationSpec(
        dtype=torch.int32,
        quant_min=torch.iinfo(torch.int32).min,
        quant_max=torch.iinfo(torch.int32).max,
        qscheme=torch.per_tensor_symmetric
        observer_or_fake_quant_ctr=MinMaxObserver.with_args(**extra_args),
    )
```

```
print(act_quantization_spec)
QuantizationSpec(dtype=torch.uint8,
observer_or_fake_quant_ctr=functools.partial(<class'torch.ao.quantization.observer.MovingAverageMinMaxObserver'>, eps=0.000244140625){},
quant_min=None, quant_max=None, qscheme=torch.per_tensor_affine, ch_axis=0, is_dynamic=False)

print(weight_quantization_spec)
QuantizationSpec(dtype=torch.int8, observer_or_fake_quant_ctr=functools.partial(<class 'torch.ao.quantization.observer.MinMaxObserver'>,
eps=0.000244140625){}, quant_min=-127, quant_max=127, qscheme=torch.per_tensor_symmetric, ch_axis=0, is_dynamic=False)

print(bias_quantization_spec)
QuantizationSpec(dtype=torch.int32, observer_or_fake_quant_ctr=functools.partial(<class 'torch.ao.quantization.observer.MinMaxObserver'>,
eps=0.000244140625){}, quant_min=-2147483648, quant_max=2147483647, qscheme=torch.per_tensor_symmetric, ch_axis=None, is_dynamic=False)
```

04
Llama Main Code Review

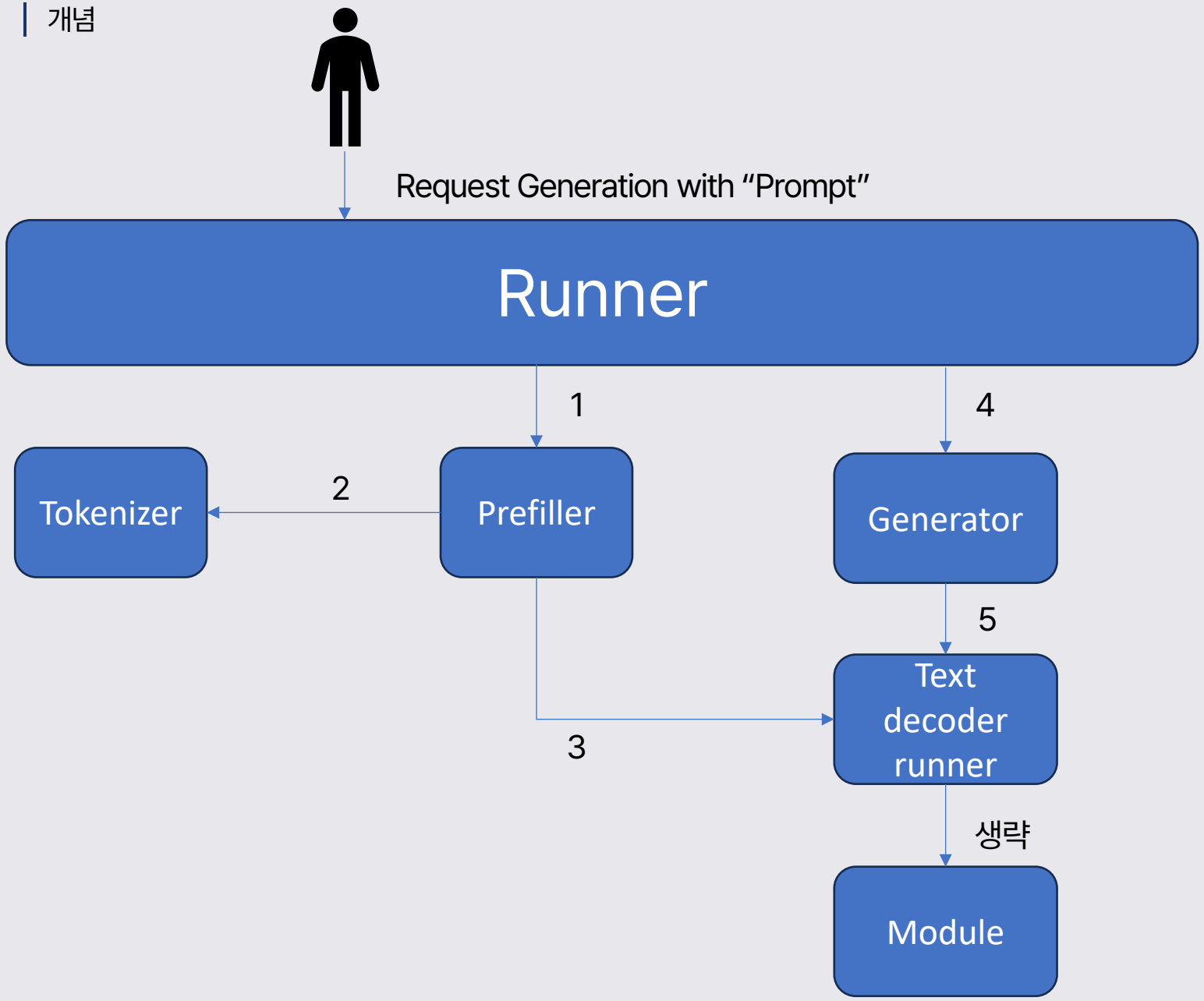
Actual implementation

개념



Actual implementation

개념



범례	설명
1	추론을 시작하기전 하나의 토큰을 미리 추론 요청
2	입력된 텍스트를 토크나이징
3	토큰을 모델에 입력 후 하나의 결과를 얻음
4	3의 결과를 추가한 뒤 ‘EOS’ 토큰이 나올 때 까지 추론 요청
5	실제 추론 수행

Actual implementation

KV-Cache?

```
Runner::Runner(const std::string& model_path,
               const std::string& tokenizer_path,
               const float        temperature)
    // NOTE: we observed ~2x loading performance increase on iPhone 15
    // and a ~5% improvement on Galaxy S22 by switching to
    // FileDataLoader instead of MmapDataLoader + UseMlockIgnoreErrors.
    : temperature_(temperature),
      module_(std::make_unique<Module>(model_path, Module::LoadMode::File)),
      tokenizer_path_(tokenizer_path), metadata_({
                                        {kEnableDynamicShape, false},
                                        {kMaxSeqLen, 128},
                                        {kUseKVCache, true},
                                        {kUseSDPAWithKVCache, false},
                                    })
{
    ET_LOG(Info,
           "Creating LLaMa runner: model_path=%s, tokenizer_path=%s",
           model_path.c_str(),
           tokenizer_path.c_str());
}
```

```
Error
Runner::load()
{
    //...
    text_decoder_runner_ = std::make_unique<llm::TextDecoderRunner>(module_.get(), //
                                                                    metadata_.at(kUseKVCache),
                                                                    metadata_.at(kVocabSize),
                                                                    temperature_);
    text_prefiller_      = std::make_unique<llm::TextPrefiller>(text_decoder_runner_.get(), //
                                                                metadata_.at(kUseKVCache),
                                                                metadata_.at(kEnableDynamicShape));

    text_token_generator_ = std::make_unique<llm::TextTokenGenerator>(tokenizer_.get(),
                                                                      text_decoder_runner_.get(),
                                                                      metadata_.at(kUseKVCache),
                                                                      std::move(eos_ids),
                                                                      &stats_);
    //...
}
```

```
::executorch::runtime::Result<exec_aten::Tensor>
TextDecoderRunner::step(TensorPtr& tokens, TensorPtr& start_pos)
{
    // ET_LOG(Info, "Input token %" PRIu64, input_token);
    if (use_kv_cache_)
    {
        auto outputs_res = module_->forward({tokens, start_pos});
        ET_CHECK_OK_OR_RETURN_ERROR(outputs_res.error());
        ET_CHECK_MSG(outputs_res.get().size() == 1,
                     "More then one output returned from executing LLM.");
        ET_CHECK_MSG(outputs_res.get()[0].isTensor(),
                     "Non Tensor Output returned from executing LLM");

        // Return the logits tensor
        return outputs_res.get()[0].toTensor();
    }
    else
    {                           // no kv cache
        (void)start_pos; // unused

        auto outputs_res = module_->forward(tokens);
        ET_CHECK_OK_OR_RETURN_ERROR(outputs_res.error());
        ET_CHECK_MSG(outputs_res.get().size() == 1,
                     "More then one output returned from executing LLM.");
    }
}
```

Actual implementation

KV-Cache!

```
Tensor& sdpa_with_kv_cache_out(KernelRuntimeContext&   ctx,
                               const Tensor&           q_projected,
                               const Tensor&           k_projected,
                               const Tensor&           v_projected,
                               Tensor&                 key_cache,
                               Tensor&                 value_cache,
                               const int64_t           start_pos,
                               const int64_t           seq_len,
                               const optional<Tensor>& attn_mask,
                               const double            dropout_p,
                               const bool              is_causal,
                               // @lint-ignore CLANGTIDY facebook-hte-ParameterMightThrowOnCopy
                               const optional<double> scale,
                               Tensor&                output);
```

변환 된 모델 어딘가에 존재함..

End.